Contents


```
clc, clear, clf, close all;
```

Analysis Setup

```
drop_range = [0 0.5 1];
est_est_range = 1;
tree_flag = 0;
tree_range = 0;%[0:0.005:0.195 0.2:0.05:0.5];

% drop_range = [0:0.02:0.38 0.4:0.03:1]; %0:0.05:1;
% est_est_range = 0:1;
% tree_flag = 0;
% tree_range = 0;
```

Main Simulation Loop

```
for est_est_flag = est_est_range
    % Initialization
    setup_agents;

    % Main Simulation Loop Setup
    % Loop setup - |trajData| has about 142 seconds of recorded data.
    secondsToSimulate = 50; % simulate about 50 seconds
    numsamples = secondsToSimulate*fs_imu;
    rand_vec = rand(secondsToSimulate*2,1);

    loopBound = floor(numsamples);
    loopBound = floor(loopBound/fs_imu)*fs_imu; % ensure enough IMU Samples

    prev_state_1_2 = ekf_1.State(5:10);
    prev_state_1_3 = ekf_1.State(5:10);

    % Log data for final metric computation.
    pqorient_1 = quaternion.zeros(loopBound,1,length(drop_range)*length(tree_range));
    pqpos_1 = zeros(loopBound,3,length(drop_range)*length(tree_range));
    pqorient_2 = quaternion.zeros(loopBound,1,length(drop_range)*length(tree_range));
    pqpos_2 = zeros(loopBound,3,length(drop_range)*length(tree_range));
    pqorient_3 = quaternion.zeros(loopBound,1,length(drop_range)*length(tree_range));
    pqpos_3 = zeros(loopBound,3,length(drop_range)*length(tree_range));
    for tree_percent = tree_range
        tree_percent;
        drop_count = 1;

        num_sec_trees = tree_percent*secondsToSimulate;
        num_sec_clear = (1-tree_percent)*secondsToSimulate;
        d_signal = rand(1,num_sec_trees*2); %disturbing signal simulates signal passing through obstabcles
        s_signal = ones(1,num_sec_clear*2); %stable signal simulates signal perfectly transimitted
        signal = [d_signal s_signal];

        for drop_percent = drop_range%(0:100)/100
            drop_percent
            fcnt = 1;
            sec_count = 1;
            while(fcnt <=loopBound)
                for ff=1:fs_imu
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    % |predict| loop at IMU update frequency.
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

                    % Simulate the IMU data from the current pose.
                    [accel_1, gyro_1, mag_1] = imu_1(trajAcc_1(fcnt,:), trajAngVel_1(fcnt, :),trajOrient_1(fcnt));
                    [accel_2, gyro_2, mag_2] = imu_2(trajAcc_2(fcnt,:), trajAngVel_2(fcnt, :),trajOrient_2(fcnt));
                    [accel_3, gyro_3, mag_3] = imu_3(trajAcc_3(fcnt,:), trajAngVel_3(fcnt, :),trajOrient_3(fcnt));

                    % Use the |predict| method to estimate the filter state based
                    % on the simulated accelerometer and gyroscope signals.
                    predict(ekf_1, accel_1, gyro_1);
                    predict(ekf_2, accel_2, gyro_2);
                    predict(ekf_3, accel_3, gyro_3);

                    % Acquire the current estimate of the filter states.
                    [fusedPos_1, fusedOrient_1] = pose(ekf_1);
                    [fusedPos_2, fusedOrient_2] = pose(ekf_2);
                    [fusedPos_3, fusedOrient_3] = pose(ekf_3);

                    % Save the position and orientation for post processing.
                    pqorient_1(fcnt,drop_count) = fusedOrient_1;
                    pqpos_1(fcnt,:,drop_count) = fusedPos_1;
```

```
                pqorient_2(fcnt,drop_count) = fusedOrient_2;
                pqpos_2(fcnt,:,drop_count) = fusedPos_2;
                pqorient_3(fcnt,drop_count) = fusedOrient_3;
                pqpos_3(fcnt,:,drop_count) = fusedPos_3;

                fcnt = fcnt + 1;
            end
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % This next step happens at the GPS sample rate.
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            % Simulate the GPS output based on the current pose.
            [lla_1, gpsvel_1] = gps(trajPos_1(fcnt,:), trajVel_1(fcnt,:) );

            % Simulate the radar measurements based on the current pose
            [rel_pos_NED_1_to_2, rel_Cov_2] = rangeMeasAddedNoise(ekf_1, trajPos_2(fcnt,:), noiseMean, noiseVar, 2);
            [rel_pos_NED_1_to_3, rel_Cov_3] = rangeMeasAddedNoise(ekf_1, trajPos_3(fcnt,:), noiseMean, noiseVar, 1000);

            % Correct the filter states based on the GPS measurement.
            fusegps(ekf_1, lla_1, Rpos, gpsvel_1, Rvel);

            % Correct the filter states based on the relative position measurement.
            prev_state_est_1_2 = fuserange(ekf_1, ekf_2, rel_pos_NED_1_to_2, rel_Cov_2, prev_state_1_2, 1/fs_gps, drop_percent, est_est_flag, rand_vec(sec_count
            prev_state_est_1_3 = fuserange(ekf_1, ekf_3, rel_pos_NED_1_to_3, rel_Cov_3, prev_state_1_3, 1/fs_gps, drop_percent, est_est_flag, rand_vec(sec_count

            % Correct the filter states based on the magnetic field measurement.
            fusemag(ekf_1, mag_1, Rmag);
            fusemag(ekf_2, mag_2, Rmag);
            fusemag(ekf_3, mag_3, Rmag);


            if ~tree_flag
                if rand_vec(sec_count) > drop_percent
                    prev_state_1_2 = ekf_1.State(5:10);
                elseif est_est_flag == 1
                    prev_state_1_2 = prev_state_est_1_2;
                end

                if rand_vec(sec_count+1) > drop_percent
                    prev_state_1_3 = ekf_1.State(5:10);
                elseif est_est_flag == 1
                    prev_state_1_3 = prev_state_est_1_3;
                end
            else
                if signal(sec_count) > drop_percent
                    prev_state_1_2 = ekf_1.State(5:10);
                elseif est_est_flag == 1
                    prev_state_1_2 = prev_state_est_1_2;
                end

                if signal(sec_count+1) > drop_percent
                    prev_state_1_3 = ekf_1.State(5:10);
                elseif est_est_flag == 1
                    prev_state_1_3 = prev_state_est_1_3;
                end
            end
            sec_count = sec_count+2;
        end
        drop_count = drop_count+1;
    end
end
for j = 1:length(drop_range)
    posd_1(:,:,j) = pqpos_1(1:loopBound,:,j) - trajPos_1( 1:loopBound, :);
    posd_2(:,:,j) = pqpos_2(1:loopBound,:,j) - trajPos_2( 1:loopBound, :);
    posd_3(:,:,j) = pqpos_3(1:loopBound,:,j) - trajPos_3( 1:loopBound, :);

    error_sum_1(:,j) = sqrt(posd_1(:,1,j).^2 + posd_1(:,2,j).^2 + posd_1(:,3,j).^2);
    error_sum_2(:,j) = sqrt(posd_2(:,1,j).^2 + posd_2(:,2,j).^2 + posd_2(:,3,j).^2);
    error_sum_3(:,j) = sqrt(posd_3(:,1,j).^2 + posd_3(:,2,j).^2 + posd_3(:,3,j).^2);
    errorTotal(j,:,est_est_flag+1) = [rms(error_sum_1(:,j)) rms(error_sum_2(:,j)) rms(error_sum_3(:,j))];
end
end
RMSE_PA = ones(length(drop_range),1).*errorTotal(1,1,1);
```

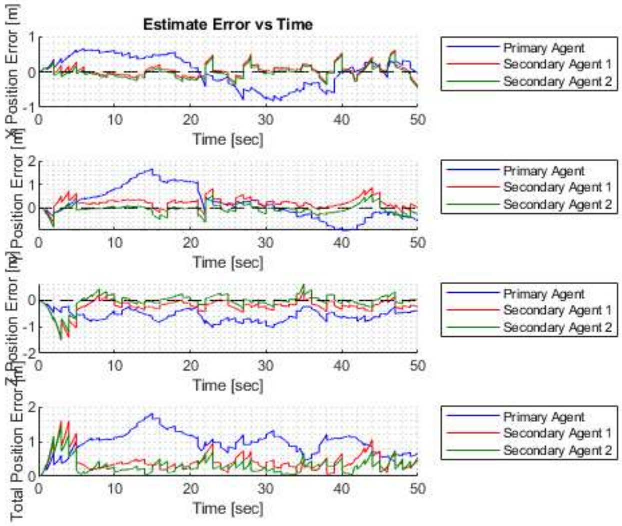
```
drop_percent =

     0
```

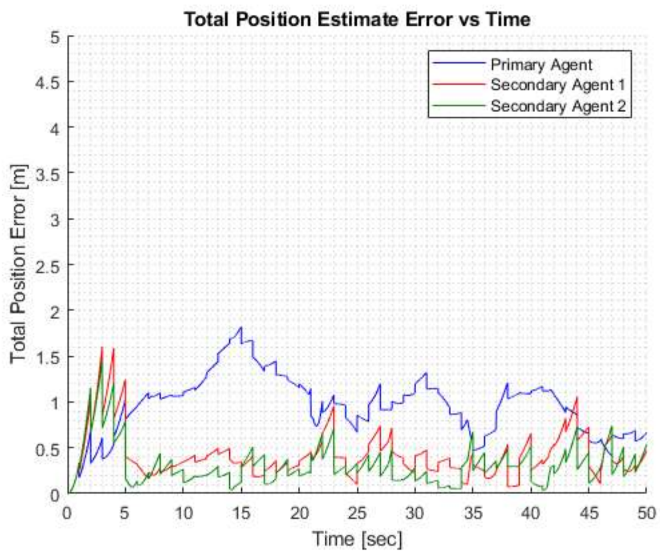
Plotting Code

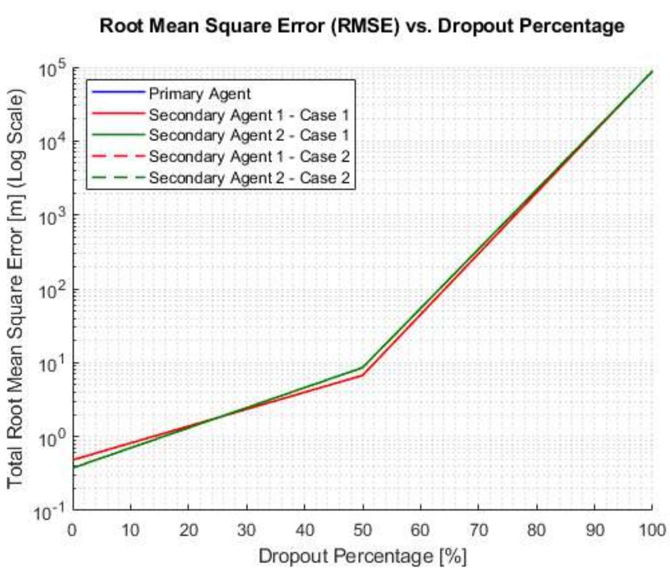
```
figure(1)
subplot(4,1,1)
title('Estimate Error vs Time')
hold on
grid minor
plot((1:loopBound)/fs_imu, posd_1(:,1,1),'b')
plot((1:loopBound)/fs_imu, posd_2(:,1,1),'r')
```

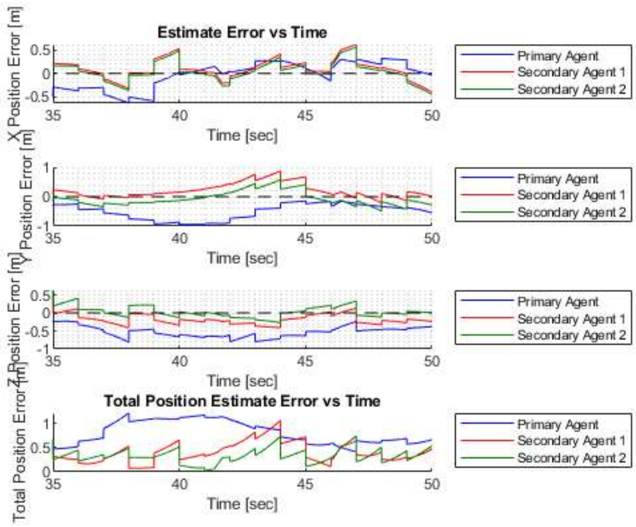
```
plot((1:loopBound)/fs_imu, posd_3(:,1,1),'Color',[0,128,0]/255)
plot((1:loopBound)/fs_imu,zeros(1,length(1:loopBound)),'k--')
xlabel('Time [sec]');ylabel('X Position Error [m]')
legend('Primary Agent', 'Secondary Agent 1', 'Secondary Agent 2','Location','northeastoutside')

subplot(4,1,2)
hold on
grid minor
plot((1:loopBound)/fs_imu, posd_1(:,2,1),'b')
plot((1:loopBound)/fs_imu, posd_2(:,2,1),'r')
plot((1:loopBound)/fs_imu, posd_3(:,2,1),'Color',[0,128,0]/255)
plot((1:loopBound)/fs_imu,zeros(1,length(1:loopBound)),'k--')
xlabel('Time [sec]');ylabel('Y Position Error [m]')
legend('Primary Agent', 'Secondary Agent 1', 'Secondary Agent 2','Location','northeastoutside')

subplot(4,1,3)
hold on
grid minor
plot((1:loopBound)/fs_imu, posd_1(:,3,1),'b')
plot((1:loopBound)/fs_imu, posd_2(:,3,1),'r')
plot((1:loopBound)/fs_imu, posd_3(:,3,1),'Color',[0,128,0]/255)
plot((1:loopBound)/fs_imu,zeros(1,length(1:loopBound)),'k--')
xlabel('Time [sec]');ylabel('Z Position Error [m]')
legend('Primary Agent', 'Secondary Agent 1', 'Secondary Agent 2','Location','northeastoutside')

subplot(4,1,4)
hold on
grid minor
plot((1:loopBound)/fs_imu, sqrt(posd_1(:,1,1).^2 + posd_1(:,2,1).^2 + posd_1(:,3,1).^2),'b')
plot((1:loopBound)/fs_imu, sqrt(posd_2(:,1,1).^2 + posd_2(:,2,1).^2 + posd_2(:,3,1).^2),'r')
plot((1:loopBound)/fs_imu, sqrt(posd_3(:,1,1).^2 + posd_3(:,2,1).^2 + posd_3(:,3,1).^2),'Color',[0,128,0]/255)
xlabel('Time [sec]');ylabel('Total Position Error [m]')
legend('Primary Agent', 'Secondary Agent 1', 'Secondary Agent 2','Location','northeastoutside')

figure(2)
title('Total Position Estimate Error vs Time')
hold on
grid minor
plot((1:loopBound)/fs_imu, sqrt(posd_1(:,1,1).^2 + posd_1(:,2,1).^2 + posd_1(:,3,1).^2),'b')
plot((1:loopBound)/fs_imu, sqrt(posd_2(:,1,1).^2 + posd_2(:,2,1).^2 + posd_2(:,3,1).^2),'r')
plot((1:loopBound)/fs_imu, sqrt(posd_3(:,1,1).^2 + posd_3(:,2,1).^2 + posd_3(:,3,1).^2),'Color',[0,128,0]/255)
xlabel('Time [sec]');ylabel('Total Position Error [m]')
legend('Primary Agent', 'Secondary Agent 1', 'Secondary Agent 2','Location','northeast')
ylim([0 5])


figure(3)
hold on; grid minor
title({'Root Mean Square Error (RMSE) vs. Dropout Percentage',''})
if (numel(size(errorTotal)) == 2)
    plot(drop_range*100, RMSE_PA,'b-','LineWidth',1.2)
    plot(drop_range*100, errorTotal(:,2,1),'r-','LineWidth',1.2)
    plot(drop_range*100, errorTotal(:,3,1),'LineStyle','-','LineWidth',1.2,'Color',[0,128,0]/255)
    set(gca,'YScale','log')
    xlabel('Dropout Percentage [%]');
    ylabel('Total Root Mean Square Error [m]')
    legend('Primary Agent', 'Secondary Agent 1', 'Secondary Agent 2','Location','northwest')
elseif (numel(size(errorTotal)) == 3)
    plot(drop_range*100, RMSE_PA,'b-','LineWidth',1.2)
    plot(drop_range*100, errorTotal(:,2,2),'r-','LineWidth',1.2)
    plot(drop_range*100, errorTotal(:,3,2),'LineStyle','-','LineWidth',1.2,'Color',[0,128,0]/255)
    xlabel('Dropout Percentage [%]');
    set(gca,'YScale','log')
    plot(drop_range*100, errorTotal(:,2,1),'r--','LineWidth',1.2)
    plot(drop_range*100, errorTotal(:,3,1),'LineStyle','--','LineWidth',1.2,'Color',[0,128,0]/255)
    ylabel('Total Root Mean Square Error [m] (Log Scale)')
    legend('Primary Agent', 'Secondary Agent 1 - Case 1', 'Secondary Agent 2 - Case 1','Secondary Agent 1 - Case 2','Secondary Agent 2 - Case 2','Location','northwe
elseif (treeflag == 1)
    plot(drop_range*100, RMSE_PA,'b-','LineWidth',1.2)
    plot(drop_range*100, errorTotal(:,2,1),'r-','LineWidth',1.2)
    plot(drop_range*100, errorTotal(:,3,1),'LineStyle','-','LineWidth',1.2,'Color',[0,128,0]/255)
    xlabel('Dropout Percentage [%]');
    set(gca,'YScale','log')
    ylabel('Total Root Mean Square Error [m] (Log Scale)')
    legend('Primary Agent', 'Secondary Agent 1', 'Secondary Agent 2')
end

figure(4)
subplot(4,1,1)
title('Estimate Error vs Time')
hold on
grid minor
plot((1:loopBound)/fs_imu, posd_1(:,1,1),'b')
plot((1:loopBound)/fs_imu, posd_2(:,1,1),'r')
plot((1:loopBound)/fs_imu, posd_3(:,1,1),'Color',[0,128,0]/255)
plot((1:loopBound)/fs_imu,zeros(1,length(1:loopBound)),'k--')
xlabel('Time [sec]');ylabel('X Position Error [m]')
legend('Primary Agent', 'Secondary Agent 1', 'Secondary Agent 2','Location','northeastoutside')
xlim([35 50]);

subplot(4,1,2)
```

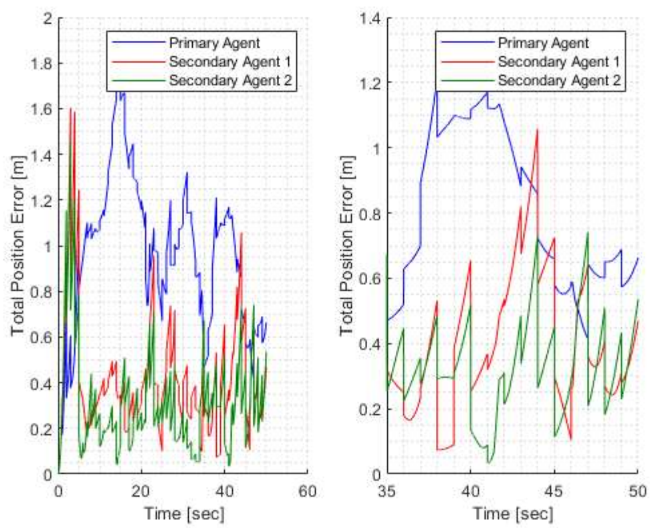
```
hold on
grid minor
plot((1:loopBound)/fs_imu, posd_1(:,2,1),'b')
plot((1:loopBound)/fs_imu, posd_2(:,2,1),'r')
plot((1:loopBound)/fs_imu, posd_3(:,2,1),'Color',[0,128,0]/255)
plot((1:loopBound)/fs_imu,zeros(1,length(1:loopBound)),'k--')
xlabel('Time [sec]');ylabel('Y Position Error [m]')
legend('Primary Agent', 'Secondary Agent 1', 'Secondary Agent 2','Location','northeastoutside')
xlim([35 50]);

subplot(4,1,3)
hold on
grid minor
plot((1:loopBound)/fs_imu, posd_1(:,3,1),'b')
plot((1:loopBound)/fs_imu, posd_2(:,3,1),'r')
plot((1:loopBound)/fs_imu, posd_3(:,3,1),'Color',[0,128,0]/255)
plot((1:loopBound)/fs_imu,zeros(1,length(1:loopBound)),'k--')
xlabel('Time [sec]');ylabel('Z Position Error [m]')
legend('Primary Agent', 'Secondary Agent 1', 'Secondary Agent 2','Location','northeastoutside')
xlim([35 50]);

subplot(4,1,4)
hold on
grid minor
plot((1:loopBound)/fs_imu, sqrt(posd_1(:,1,1).^2 + posd_1(:,2,1).^2 + posd_1(:,3,1).^2),'b')
plot((1:loopBound)/fs_imu, sqrt(posd_2(:,1,1).^2 + posd_2(:,2,1).^2 + posd_2(:,3,1).^2),'r')
plot((1:loopBound)/fs_imu, sqrt(posd_3(:,1,1).^2 + posd_3(:,2,1).^2 + posd_3(:,3,1).^2),'Color',[0,128,0]/255)
xlabel('Time [sec]');ylabel('Total Position Error [m]')
legend('Primary Agent', 'Secondary Agent 1', 'Secondary Agent 2','Location','northeastoutside')
xlim([35 50]);

figure(4)
title('Total Position Estimate Error vs Time')
hold on
grid minor
plot((1:loopBound)/fs_imu, sqrt(posd_1(:,1,1).^2 + posd_1(:,2,1).^2 + posd_1(:,3,1).^2),'b')
plot((1:loopBound)/fs_imu, sqrt(posd_2(:,1,1).^2 + posd_2(:,2,1).^2 + posd_2(:,3,1).^2),'r')
plot((1:loopBound)/fs_imu, sqrt(posd_3(:,1,1).^2 + posd_3(:,2,1).^2 + posd_3(:,3,1).^2),'Color',[0,128,0]/255)
xlabel('Time [sec]');ylabel('Total Position Error [m]')
legend('Primary Agent', 'Secondary Agent 1', 'Secondary Agent 2','Location','northeastoutside')


figure(5);
title('Total Position Estimate Error vs Time')
subplot(1,2,1)
hold on
grid minor
plot((1:loopBound)/fs_imu, sqrt(posd_1(:,1,1).^2 + posd_1(:,2,1).^2 + posd_1(:,3,1).^2),'b')
plot((1:loopBound)/fs_imu, sqrt(posd_2(:,1,1).^2 + posd_2(:,2,1).^2 + posd_2(:,3,1).^2),'r')
plot((1:loopBound)/fs_imu, sqrt(posd_3(:,1,1).^2 + posd_3(:,2,1).^2 + posd_3(:,3,1).^2),'Color',[0,128,0]/255)
xlabel('Time [sec]');ylabel('Total Position Error [m]')
legend('Primary Agent', 'Secondary Agent 1', 'Secondary Agent 2','Location','northeast')
subplot(1,2,2)
hold on
grid minor
plot((1:loopBound)/fs_imu, sqrt(posd_1(:,1,1).^2 + posd_1(:,2,1).^2 + posd_1(:,3,1).^2),'b')
plot((1:loopBound)/fs_imu, sqrt(posd_2(:,1,1).^2 + posd_2(:,2,1).^2 + posd_2(:,3,1).^2),'r')
plot((1:loopBound)/fs_imu, sqrt(posd_3(:,1,1).^2 + posd_3(:,2,1).^2 + posd_3(:,3,1).^2),'Color',[0,128,0]/255)
xlabel('Time [sec]');ylabel('Total Position Error [m]')
legend('Primary Agent', 'Secondary Agent 1', 'Secondary Agent 2','Location','northeast')
xlim([35 50]);
```


Estimate Error vs Time
X Position Error [m]
Y Position Error [m]
Z Position Error [m]
Total Position Error [m]
Time [sec]
Primary Agent
Secondary Agent 1
Secondary Agent 2


Total Position Estimate Error vs Time
Total Position Error [m]
Time [sec]
Primary Agent
Secondary Agent 1
Secondary Agent 2


Root Mean Square Error (RMSE) vs. Dropout Percentage
Total Root Mean Square Error [m] (Log Scale)
Dropout Percentage [%]
Primary Agent
Secondary Agent 1 - Case 1
Secondary Agent 2 - Case 1
Secondary Agent 1 - Case 2
Secondary Agent 2 - Case 2





All User-defined Functions

```
function prev_state_est = fuserange(ekf_1, ekf_n, rel_vec, rel_Cov, prev_state_1, dt, drop_percent, est_est_flag, rand_num, tree_flag, signal)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function takes the EKF object for Agent n and Agent 1 as input.
% Processes the relative position measurement from radar with the estimated
% position of Agent 1 to compute the measured position of Agent n. This
% position measurement of Agent n is fused into the EKF estimate for Agent
% n to correct for the drift of the IMU sensors
%
% Author: Nathan Gurgens
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Compute the position of Agent n given the relative position measurement
% between Agent 1 and Agent n and the estimated position of Agent 1

[pos_meas_n, meas_Cov_n, prev_state_est] = compute_pos(ekf_1, rel_vec, rel_Cov, prev_state_1, dt, drop_percent, est_est_flag, rand_num, tree_flag, signal);

if ~isnan(pos_meas_n)
    range_correction(ekf_n, pos_meas_n, @rangeMeasFcn, meas_Cov_n, @rangeMeasJacobianFcn)
end

end


function range_correction(ekf, z, measFcn, measCov, measJacobianFcn)

x_est = ekf.State;
P_est = ekf.StateCovariance;
h = measFcn(x_est);
H = measJacobianFcn();

K = P_est*H.'*inv(H*P_est*(H.') + measCov);
x_est = x_est + K*(z - h);
P_est = P_est - K*H*P_est;
```

```matlab
% x_est = repairQuaternion(ekf,x_est);

ekf.State = x_est;
ekf.StateCovariance = P_est;

end

function z = rangeMeasFcn(x)

% Getting only 3 positional terms from state x

z = [x(5) x(6) x(7)].';

end

function dhdx = rangeMeasJacobianFcn()

dhdx = [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
        0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
        0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];

end

%Copied from MATLAB's BasicEKF.m to avoid syntax issues
function x = repairQuaternion(obj, x)
    % Normalize quaternion and enforce a positive angle of
    % rotation. Avoid constructing a quaternion here. Just do the
    % math inline.

    % Used cached OrientationIdx
    idx = obj.OrientationIdx;
    qparts = x(idx);
    n = sqrt(sum(qparts.^2));
    qparts = qparts./n;
    if qparts(1) < 0
        x(idx) = -qparts;
    else
        x(idx) = qparts;
    end
end

function [rangeMeas,covariance_n] = rangeMeasAddedNoise(ekf, trajPos_n, mean, variance, seedInitializer)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function takes the reference trajectories for Agents in the
% formation and adds white Gaussian noise to simulate the measured position
% information of the agents. The simulated measurements of the agents are
% then used to calculate the range vector between two agents for IMU/GPS
% fusion among the Agents within the formation.
%
% Author: Malav Naik
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% rng(agentID);
% for ii = 1:numel(trajPos_1(1,:))
%     rng(ii);
%     noise_1(ii,1) = mean + sqrt(gps_variance)*randn(size(trajPos_1(ii))); % Variance here needs to be GPS variance
%     trajPos1_meas(ii,1) = trajPos_1(ii) + noise_1(ii);
% end

trajPos1_meas = pose(ekf).';

% trajPos1_meas
% covariance_1 = variance*eye(numel(trajPos_1(1,:)));
% rng(agentID+5);
for jj = 1:numel(trajPos_n(1,:))
    rng(jj+seedInitializer);
    noise_n(jj,1) = mean + sqrt(variance(jj))*randn(size(trajPos_n(jj)));
    trajPos_n_meas(jj,1) = trajPos_n(jj) + noise_n(jj);
end
% trajPos_n_meas
covariance_n = diag(variance); %*eye(numel(trajPos_n(1,:)));

rangeMeas = trajPos_n_meas - trajPos1_meas;

end

function [pos_meas_n, meas_Cov_n, prev_state_est] = compute_pos(ekf, rel_vec, rel_Cov, prev_state_1, dt, drop_percent, est_est_flag, rand_num, tree_flag, signal)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function takes the state estimate of Agent 1, the range measurement
% between Agent 1 and other agents in the formation and the range
% measurement's covariance matrix to detemine the position measurement
% for Agents in the formation.
%
% Author: Malav Naik
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
curr_pos_1 = ekf.State(5:7);
prev_pos_1 = prev_state_1(1:3);
prev_vel_1 = prev_state_1(4:6);
if ~tree_flag
    [agent1State_est_est, prev_state_est] = stateComputeAgent1(prev_pos_1, prev_vel_1, dt, curr_pos_1, drop_percent, est_est_flag, rand_num);
else
    [agent1State_est_est, prev_state_est] = trees_analysis(prev_pos_1,prev_vel_1,dt,curr_pos_1,drop_percent,signal,est_est_flag);
```

```
end

if ~isnan(agent1State_est_est)
    agent1Pos_est_est(1:3,1) = agent1State_est_est(1:3);

    pos_meas_n = agent1Pos_est_est + rel_vec;

    agent1Pos_est_est_Cov = ekf.StateCovariance;
    meas_Cov_n = rel_Cov + agent1Pos_est_est_Cov(5:7,5:7);
else
    pos_meas_n = NaN;
    meas_Cov_n = NaN;
end

end

function [pos_est_est_1, prev_state_est] = stateComputeAgent1(prev_pos_1,prev_vel_1,dt,curr_pos_1,drop_percent,est_est_flag, rand_num)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% the loop randomly generates random number between 0 and 1, and omit
% signal that is less than 50% and returns the outcome of the estimate of
% the previos one (est_est),if the signal transmitted is greater than 50%,
% then we proceed with the original pos
%
% Author: Zhiyao Song
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
rand_num;
if est_est_flag == 1
    if any(rand_num < drop_percent)
        pos_est_est_1 = prev_pos_1 + prev_vel_1*dt;
        prev_state_est = [pos_est_est_1 prev_vel_1].';
    else
         pos_est_est_1 = curr_pos_1;
         prev_state_est = NaN;
    end
else
    if any(rand_num < drop_percent)
        pos_est_est_1 = NaN;
        prev_state_est = NaN;
    else
         pos_est_est_1 = curr_pos_1;
         prev_state_est = NaN;
    end

end

end

function [pos_est_est_1, prev_state_est] = trees_analysis(prev_pos_1,prev_vel_1,dt,curr_pos_1,drop_percent,signal,est_est_flag)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function generates the tree analysis results.
%
% Author: Zhiyao Song
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if est_est_flag == 1
    if any(signal < drop_percent)
        pos_est_est_1 = prev_pos_1 + prev_vel_1*dt;
        prev_state_est = [pos_est_est_1 prev_vel_1].';
    else
         pos_est_est_1 = curr_pos_1; %pose(ekf_1)
         prev_state_est = NaN;
    end
else
    if any(signal < drop_percent)
        pos_est_est_1 = NaN;
        prev_state_est = NaN;
    else
         pos_est_est_1 = curr_pos_1;
         prev_state_est = NaN;
    end
end
end
```

```
drop_percent =

    0.5000


drop_percent =

     1
```